Game-Theoretic Framework for Integrity Verification in Computation Outsourcing

Qiang Tang and Balázs Pejó

SnT, University of Luxembourg
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
{qiang.tang, balazs.pejo}@uni.lu

Abstract. In the cloud computing era, in order to avoid computational burdens, many organizations tend to outsource their computations to third-party cloud servers. In order to protect service quality, the integrity of computation results need to be guaranteed. In this paper, we develop a game theoretic framework which helps the outsourcer to minimize its cost while ensuring the integrity of the outsourced computation. We then apply the proposed framework to two collaborative filtering algorithms and demonstrate the equilibriums together with the corresponding minimal costs. Finally, we show that, by including the intermediate results in the final output, further cost reduction can be achieved.

1 Introduction

In today's data-centric world, all the companies collect as much data as possible from their customers in order to provide better services, e.g. in a form of personalization. However, processing the collected data is often very computation-intensive, making it infeasible for companies without the necessary resources. In the cloud computing era, a natural solution is to outsource these computations to a cloud service provider. In such a case, two issues arise. One is about the integrity of the computed results. The cloud server may provide some fake results instead of spending its own resources in computing the correct ones. Motivations behind such misbehavior could differ, but saving its own cost and deliberately disrupting the service are two obvious ones that we foresee. The other is confidentiality, or more generally privacy issues. Despite of its importance, this is not tackled in this paper.

1.1 Problem Statement

As a standard practice, we refer to the outsourcer as *client* and the outsourcee as *server*. For simplicity, we assume that an outsourced computation produces K outputs, which can be checked by the client individually. For example, in the case of recommender algorithms [6], the K outputs are the prediction results for the end users. Based on this assumption, without loss of generality, we consider the following cheating and verification strategies.

– *Cheating strategy*: The server sets ρ percent of the K results to be random numbers. The parameter ρ is referred to as the cheating rate.
– *Verification strategy*: The client chooses σ percent of the K outputs to verify, by recomputing them. The parameter σ is referred to as the checking rate.

As such, the detection rate P_d (i.e. the probability that the client detects the server's cheating) can be computed as follows.

$$P_d(K, \sigma, \rho) = 1 - \frac{\binom{(1-\rho)K}{\sigma K}}{\binom{K}{\sigma K}} \tag{1}$$

In practice, the cheating rate ρ is chosen by the server and unknown to the client. This immediately leads to a question: how the client should set the checking rate σ to achieve an acceptable integrity guarantee? A more general question is: how the client can decide how much it should pay to the server when cheating is (not) detected? We try to answer these questions in this paper.

1.2 Related Work

Data-mining-as-a-service (DMAS) [4] has been proposed to enable clients with insufficient computing resources to mine large volumes of data through outsourcing to cloud servers. One of the application scenarios is recommender systems, in particular collaborative filtering systems. In this case, a recommender service provider may collect a large amount of data from its customers and need to generate meaningful predictions for them. Due to the computational complexity, the recommender service provider may outsource the computations to a third-party cloud server. For example, Netflix outsources its computations to Amazon.

By the popularity gain of outsourcing, serious security challenges emerged such as confidentiality and integrity [9]. Result integrity is usually achieved by some kind of verification mechanism. For integrity verification, [10] proposed a solution leveraging on artificial items. In [1], the outsourced results are verified by constructing a set of items from real ones and using these as evidence to check mining results. In [8], the authors develop a game theoretic framework to improve the existing verification mechanism. In [7], the authors present experimental results with respect to the verification methods from [8].

1.3 Our Contribution

Referring to Equation (1), the cheating rate ρ is unknown to the client so it cannot calculate P_d. To tackle this problem, we require the client to define a threshold cheating toleration rate θ. By doing so, the client ensures that if the server sets the cheating rate above this threshold ($\rho \geq \theta$) then it will be caught at least with probability $P_d(K, \sigma, \theta)$. Subsequently, the client can set other parameters in an optimal manner. Our contribution can be summarized as follows.

– We define a two-player Stackelberg game, where the client wants to outsource some computation to the server and verify the results. We provide a strategy for the client to minimize its cost and force a rational server to execute the computation task honestly, i.e. *not-cheat* behavior is the dominant strategy for the server.
– We apply the framework to two collaborative filtering recommendation algorithms and show experimental results with respect to well-known Movielens and Netflix data sets. We propose some modification on the outsourced algorithms' output to further reduce the client's cost.

1.4 Organization

The paper is organized as follows. Section 2 introduces preliminary on recommender systems and game theory. Section 3 recaps an existing game theoretic framework for integrity verification from [8]. Section 4 introduces a new game theoretic framework. Section 5 contains two use cases for the framework. The conclusion and future works are in Section 6.

2 Preliminaries

This section contains an introduction to recommendation systems and some basic notions of game theory.

2.1 Recommender System

The two of most popular collaborative filtering approaches are the neighborhood methods and latent factor models [6]. Neighborhood methods center on computing the relationships between items (or users). Latent factor models try to explain the ratings by characterizing both items and users on some factors inferred from the rating patterns. Typically, latent factor models are carried out via matrix factorization (MF) [2].

In a recommender system, the item set is denoted by $\mathbf{I} = (1, 2, \ldots, I)$ and the user set is denoted by $\mathbf{U} = \{1, 2, \ldots, U\}$. A user u's ratings are denoted by a vector $\mathbf{R}_u = (r_{u,1}, \ldots, r_{u,I})$. The ratings are often an integer between 1 and 5. If user u has not rated item i then $r_{u,i} = 0$. The ratings are often organized in a rating matrix $\mathbf{R}$, where $\mathbf{R}_u$ forms the u-th row. With respect to $\mathbf{R}_u$, a binary vector $\mathbf{Q}_u = (q_{u,1}, \ldots, q_{u,I})$ is defined as follows: $q_{u,i} = 1$ if $r_{u,i} \neq 0$ and $q_{u,i} = 0$ otherwise. Basically, $\mathbf{Q}_u$ indicates which items have been rated by user u. Let $N = \sum_u \sum_i q_{u,i}$, which is the total number of ratings.

In the literature a lot of recommender algorithms have been proposed. To illustrate our framework, we consider two collaborative filtering methods: one from neighborhood based and one from latent factor models.

Weighted Slope One Algorithm The item-based collaborative filtering algorithm Weighted Slope One [3] exploits deviation metrics with popularity differential notion between items. It predicts the rating of an item for a user from a pair-wise deviations of item ratings. The algorithm has two stages: the computation stage and the prediction stage.

Computation stage. In this stage, two matrices $\Phi_{I\times I}$ and $\Delta_{I\times I}$ are generated. For every $1 \leq i, j \leq I$, $\phi_{i,j}$ and $\delta_{i,j}$ are defined in Equation (2). In more detail, $\phi_{i,j}$ is the number of users who rated both item i and item j, while $\delta_{i,j}$ is the deviation of the ratings between item i and item j.

$$\phi_{i,j} = \sum_{u=1}^{U} q_{u,i}q_{u,j}, \qquad \delta_{i,j} = \sum_{u=1}^{U} q_{u,i}q_{u,j}(r_{u,i} - r_{u,j}) \tag{2}$$

Prediction stage. This stage uses both Φ and Δ as well as the original rating matrix $\mathbf{R}$ to predict the unrated ratings. To compute the predictions for user u with respect to item i, the formula is as follows.

$$p_{u,i} = \frac{\sum_{j\in\mathbf{I}/\{i\}} \delta_{i,j} + r_{u,j}\phi_{i,j}}{\sum_{j\in\mathbf{I}/\{i\}} \phi_{i,j}} \tag{3}$$

Matrix Factorization MF models [2] map both users and items to a joint latent factor space with dimension $c \in \mathbb{Z}^+$. Each item i (user u) is associated with a vector $s_i \in \mathbb{R}^c$ ($t_u \in \mathbb{R}^c$). s_i measures the extent to which the item possesses those factors, while t_u measures the interest of the user with respect to those factors. The inner product $s_i \cdot t_u = \sum_{k=1}^{c} s_{i_k} t_{u_k}$ captures the user u's overall interest in item i, i.e. its potential rating $\hat{r}_{u,i}$. With these notions, to get appropriate predictions, Equation (4) should be minimized, where the parameter γ is to control the extent of regularization, Ψ is the set of pairs (u, i) where $r_{u,i} \neq 0$ and $e_{u,i} = r_{u,i} - \hat{r}_{u,i}$.

$$\min_{s^*,t^*} \sum_{(u,i)\in\Psi} e_{u,i}^2 + \gamma(||s_i||^2 + ||t_u||^2) \tag{4}$$

Equation (4) can be minimized in many ways, we present here one widely used machine learning technique.

Stochastic gradient descent. SGD is a stochastic approximation of the gradient descent (GD) optimization method, which finds the local minimum of a function using gradient descent. GD takes steps proportional to the negative of the gradient of the function at the current point. In SGD, the real gradient is approximated by a gradient at a single example. The updating process is shown in Equation (5), where ϵ is the learning rate.

$$s_i \leftarrow s_i + \epsilon(2e_{u,i}t_u - \gamma s_i), \quad t_u \leftarrow t_u + \epsilon(2e_{u,i}s_i - \gamma t_u) \tag{5}$$

2.2 Basic Notion of Game Theory

Game theory [5] provides a formal approach to model situations where players must choose optimal actions considering the mutual effects of other players' decisions. A normal form game can be written as $G = \{n, (X_i, \pi_i(\cdot))_{i=1\ldots n})\}$, where n is the number of players, X_i is a set of actions for player i and $\pi(x_i, x_{-i})$ is the payoff function where $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. $x_i \in X_i$ is called a best response action for player i to his rivals' actions x_{-i} if $\pi(x_i, x_{-i}) \geq \pi(x_i', x_{-i})$ for all $x_i' \in X_i$. A strategy x_i is dominant for player i, if it is always better than any other strategy, for any profile of other players' actions. An action profile x^* is a pure Nash equilibrium of the game G if and only if x_i^* is best response action to x_{-i}^* for all $i \in \{1, \ldots, n\}$. In other words, equilibrium is when no player can do better by unilaterally changing his strategy.

One special kind of games is called Stackelberg game, in which the players are not making their decisions simultaneously: instead there is a leader who exposes its strategy first, then the followers act. This is a game of perfect information, where all players know which decision node they are in, e.g. the follower knows the leader's decision. These games can be solved by a backward induction procedure. This basically means the following: it is a common knowledge what rational followers would do in each possible situation, so the leader chooses that particular one where the corresponding rational answer has the highest payoff for himself.

3 Existing Game Theoretic Model

In the paper, we use the notations shown in Table 1.

Name	Meaning	Assumption
K	Number of outputs	$0 < K$
C	Cost of computation	$C = 1$
W	Payment for the server when cheating is not detected	$W = 1 + w$
X	Payoff for the server when cheating is detected	$X \leq W - C$
F	Punishment when cheating is detected	$F \geq 0$
ρ	Cheating rate	$0 \leq \rho \leq 1$
σ	Checking rate	$0 < \sigma < 1$
P_d	Detection rate	$0 \leq P_d \leq 1$
θ	Tolerated cheating rate	$0 < \theta < 1$
B	Benefit for the client in case of no cheating	$B = 1 + b$
V_d	Cost of verification	$0 < V_d < 1$
λ	Unknown benefit reducer	$\lambda \geq 0$
κ	Known benefit reducer	$\kappa = B$

Table 1: List of Parameters

The benefit reducers (i.e. λ and κ) are the loss factors for the client if the server cheats. In other words, if the cheating is caught, the benefit for the client is $B - \kappa$, while if the cheating is undetected it is $B - \lambda$. The rest of the variables defined above are straightforward.

Next, we first recap the work from [8] and show the problem with it. Then, we describe some assumptions (shown in the of **Assumption** column of Table 1) for our new framework.

3.1 An Existing Model

In [8], Vaidya et al. introduced a game-theoretic approach which can be used on top of a verification mechanism for the outsourcing scenario. The proposed payoff matrix for the server is shown in Table 2.

	cheat	*not-cheat*
Detected	X	$W - C$
Not-detected	$W - (1 - \rho)C$	$W - C$

Table 2: Payoff Matrix

The payoff is basically the payment minus the cost of executed calculation. The goal is to prevent the server from cheating by setting the parameters in a way that *not-cheat* is a dominant strategy. Given a detection rate P_d, Vaidya et al. showed that it can be established when Inequality (6) is true.

$$W - C \geq (1 - P_d)(W - (1 - \rho)C) + P_d X \quad \Rightarrow \quad P_d \geq \frac{\rho C}{W - X - (1 - \rho)C} \tag{6}$$

Our first observation is the lack of the other player (client) in the model. It means that this is a decision model for the server where the parameters W, C and X are constant. However, to model the interaction between the client and the server, the client has to have choices as well.

Our second observation is about the payoff in case of detected cheating. The authors in [8] inappropriately used a single variable X. Instead, this case should be represented by a "cost" and a "payment" just like in the rest of the payoff matrix. In particular, the "cost" is actually the computation cost reduced by the cheating, i.e. $(1 - \rho)C$, while the "payment" is rather a punishment F because the cheating was caught.

3.2 Further Assumptions

To simplify our discussion, we make some rational assumptions: C is the cost of carrying out the computations, so it is normally known by both parties. If we assume that it is common knowledge, we can normalize it to be $C = 1$ and use it to define other parameters. With this simplification, we can set the payment from the client to the server $W = 1 + w$ where $w \geq 0$. *We call w as fee in the rest of the paper.* In a similar manner, we define the income b where $B = 1 + b$

and $b \geq w$. The cost of verification V_d is a function of the checking rate σ. We have $V_d : [0,1] \to [0,1]$, since C is normalized. It is obvious that $V_d(0) = 0$ and $V_d(1) = 1$.

The punishment F is the amount that the server must pay to the client if the cheating is detected. It compensates for the client's loss: in case of detected cheating, any result can be random, so the output must be discarded, i.e. all the benefit is lost, so the known benefit reducer $\kappa = B$. On the other hand, if the cheating is undetected, there is neither punishment nor known benefit loss. However, unknown benefit loss does occur, which means $\lambda \geq 0$.

4 New Game Theoretic Framework

We define a new payoff matrix and a Stackelberg game, where the client can set the parameters in a way that *not-cheat* will be a dominant strategy for the server.

4.1 Our Payoff Matrix

The new payoff matrix is shown in Table 3. If there is no cheating, the payoff for the client is its own cost (fee w for the server and verification cost V_d) deducted from its income b, while the server's payoff is exactly the fee w. If there is cheating but not detected, the server's payoff grows with the cheating factor ρ, while the client's payoff decreases with the unknown benefit reducer λ. On the other hand, if it is detected, there is no fee w, rather a punishment F is enforced. Also, the client has no benefit B in this case.

		cheat	*not-cheat*
Detected	Server	$-F-(1-\rho)$	$\varnothing$
	Client	$-V_d+F$	
Not-Detected	Server	$w+\rho$	w
	Client	$b-\lambda-w-V_d$	$b-w-V_d$

Table 3: Our Payoff Matrix

We want to avoid the case when *cheat* results in more gain for the client than *not-cheat*. In other words, the client should always prefer to obtain the correct output rather than any manipulated one. However, this is not true any more when the punishment F is very high. In order to avoid this unreasonable situation, we require Inequality (7) to hold.

$$b-w-V_d \geq (1-P_d)(b-\lambda-w-V_d)+P_d(F-V_d) \;\Rightarrow\; F \leq b-w+\frac{1-P_d}{P_d}\lambda \quad (7)$$

Since λ only affects the client's payoff which is always higher in case of *not-cheat*, we can assume $\lambda = 0$. For simplicity reasons, we set $F = 0$. The explicit “punishment” will simply be the payment elimination (i.e. nothing is paid by the client), and the implicit ”punishment” will be that the client will not outsource anything to this server anymore. The downside of this setting is that, from the server’s point of view, this just makes the *cheat* more attractive. Therefore, in order to make *non-cheat* dominant, Equation (8) should hold.

$$w \geq (1-P_d)(w+\rho) + P_d(\rho-1) \quad \Rightarrow \quad \frac{\rho}{1+w} \leq \frac{1}{1+w} \leq P_d \tag{8}$$

4.2 The Stackelberg Game

In the game, the client can first make an offer $W = 1 + w$ to the server. If the server rejects this, the game terminates just as in case when no offer was made, and the payoffs are zero for both players. If the offer is accepted, then the server carries out the computation with some level of cheating. The extensive form of this game is shown in Fig. 1, where (G_c, G_s) are the payoffs for the client and for the server respectively.

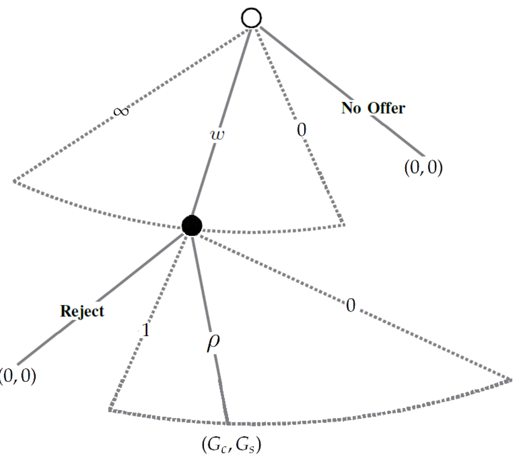

Fig. 1: The game’s extensive form

This game is a Stackelberg game, where the client is the leader and the server is the follower. Furthermore, this framework assumes that the benefit B, the number of outputs K and the threshold cheating toleration rate θ are pre-determined constants known by both parties, so the game has perfect information. The game’s normal form is shown in Table 4, where G'_c and G'_s are defined by Equation (9) and (10).

$$G'_c = -V'_d P_d(K, \sigma', \rho') + (b - w' - V'_d)(1 - P_d(K, \sigma', \rho')) \tag{9}$$

$$G'_s = (\rho' - 1)P_d(K, \sigma', \rho') + (w' + \rho')(1 - P_d(K, \sigma', \rho')) \tag{10}$$

$w \backslash \rho$	0	$\cdots$	ρ'	$\cdots$	1
0	$(0, 0)$	$\cdots$	$(0, 0)$	$\cdots$	$(0, 0)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
w'	$(b - w' - V'_d,\ w')$	$\cdots$	(G'_c, G'_s)	$\cdots$	$(-V'_d,\ 0)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$

Table 4: The game's normal form

With respect to (G'_c, G'_s), the client can only directly set w' without the knowledge of ρ'. The question is for the client to determine σ' and V'_d under the constraints defined by Inequality (7) and (8). To this end, the client can proceed as follows.

1. Each possible offer w' establishes a lower bound $(1+w')^{-1}$ for the detection rate, required by Equation (7).
2. Recall that a threshold cheating toleration rate θ should be enforced. Given w', the client can compute σ' such that $(1+w')^{-1} \leq P_d(K, \sigma', \theta)$.
3. With σ', it is easy to compute the corresponding verification cost V'_d.

In summary, for each w', there are corresponding σ' and V'_d which should be regarded as public to both parties, i.e. the game still has perfect information. This σ' ensures that *not-cheat* is the dominant strategy. For obvious reasons, $V'_d \leq b - w'$ should hold, otherwise the client's payoff would be negative. Formally, the client's goal is to minimize its cost, which is the sum of fee w and verification cost V_d. The minimization problem is defined as follows.

$$\min_w [w + V_d] \quad \text{s.t.} \quad \frac{1}{1+w} \leq P_d(K, \sigma, \theta) \quad \text{and} \quad b - w \geq V_d \tag{11}$$

4.3 A Straightforward Illustration

For illustration purpose, we ideally assume the verification cost is linear with respect to the checking rate so that we can simply have $V_d = \sigma$. While the analysis of more realistic scenarios appear in next section.

To highlight some properties of our game, we set the following parameters: $\theta = 0.0001$, $B = 1 + b = 1.1$ and $K = 1\,000\,000$. Now, the corresponding game's normal form is shown in Table 5. These numbers are calculated based on Equation (9) and (10), where the parameter σ' and V'_d are defined in a way that Inequality (8) is tight. Due to this, both players' payoffs are the highest in case of *not-cheat* ($\rho = 0$ in the table).

We can observe the following. If the cheating rate ρ is higher than the tolerated cheating rate θ (e.g. $\rho = 0.0002$), the detection rate P_d is high enough to ensure the negative payoffs for the server. While if it is less (e.g. $\rho = 0.000\,001$),

$w \backslash \rho$	0	$\cdots$	0.000 001	$\cdots$	0.0002	$\cdots$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$
0.001	(0.032,0.001)	$\cdots$	(0.026,-0.066)	$\cdots$	(-0.067,-0.9998)	$\cdots$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$
0.010	(0.045,0.01)	$\cdots$	(0.041,-0.036)	$\cdots$	(-0.045,-0.9997)	$\cdots$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$
0.100	(-0.024,0.1)	$\cdots$	(-0.024,0.074)	$\cdots$	(-0.024,-0.991)	$\cdots$

Table 5: The Normalized Payoffs

then the payoff for the server may be positive, but it is still less than the *not-cheat* case. So the only Equilibrium in this game is when the payoff for the client is maximal in case of *not-cheat*.

Next, we try to find the optimal w^* which corresponds to this maximum. To do this, we will minimize the cost for the client as in Inequality (11). For this minimization, setting the income b is not necessary, however, if the calculated minimal cost is higher than b, then the client will not make any offer at all. So we only need the tolerated cheating rate θ and the number of outputs K. In Table 6 we varied these two to show the corresponding minimal costs, which consist of the optimal fee w^* and its verification cost V_d^*.

θ	$V_d^* = \sigma^*$	w^*
0.001 00	0.006 89	0.000 99
0.000 10	0.045 64	0.009 45
0.000 01	0.237 46	0.071 20

(a) $K = 1\,000\,000$

θ	$V_d^* = \sigma^*$	w^*
0.001 00	0.000 12	0.000 01
0.000 10	0.000 92	0.000 10
0.000 01	0.006 89	0.000 99

(b) $K = 100\,000\,000$

Table 6: Optimal Parameters

The minimization here and through the paper were done via exhaustive search using the *Mathematica* software. $\{0.000\,01, 0.000\,02, \dots\}$ is used as a default set of the possible values of w if not stated otherwise. A similar method is always usable, however, with higher level of discretization, the resulted minimum is possibly different from the actual one. On the other hand, the client does not need to find the equilibrium (i.e. the exact fee corresponding to the minimal cost), rather a viable solution, i.e. finding a cost which is less then the income b, so this method is a realistic solution in practice.

The difference between Table 6a and 6b shows, that with larger output size (e.g. bigger dataset), less cost can be achieved (i.e. $w^* + V_d^*$ decrease when K increase). We emphasize that the cost values are normalized. Table 6b's numbers are correspond to 100 times bigger output-size than Table 6a, so lets assume its computation cost is also 100 times more. This actually means, even if the normalized cost values $(w^* + V_d^*)$ in Table 6b are smaller than those in Table 6a, they will be larger in reality: multiplying Table 6b with 100 would result in 64%, 85% and 155% more computation cost for $\theta = \{0.001, 0.0001, 0.000\,01\}$ respectively. This table also shows, that even in the ideal case, the verification cost V_d^* is the more than the fee w^*.

5 Real-world Use Cases

The outsourced algorithms usually contain more stages or iterations, where the nth stage uses the outputs of the $(n-1)$th stage. An general form of such a multi-stage structure is shown in Fig. 2.

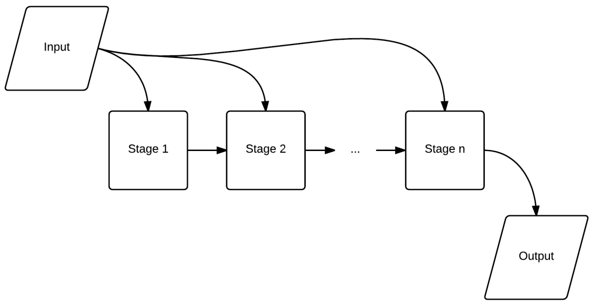

Fig. 2: Multi-stage Computations

This effect can easily be shown in Weighted Slope One, described in Section 2. In order to calculate a prediction $p_{u,i}$, besides the prediction stage calculation one must calculate the corresponding ith row in Δ and Φ (see Section 2). Now, if we want to calculate $p_{u',i'}$, we either must calculate another row in Δ (and Φ) (i.e. $i' \neq i$), or we are lucky and this row has been already calculated (i.e. $i' = i$). The more predictions we want, the more likely that we already have the corresponding rows. Furthermore, as soon as we have at least one prediction for each item i, the whole two matrices are known, so only prediction stage calculations must be computed for the remaining predictions.

In our experiments, we chose the MovieLens (ML) and the Netflix datasets, shown in Table 7. In the Weighted Slope One use case we used[1] value-based implementation in python to highlight the relationship between the stages's outputs element-wise. For this purpose, we used the two smaller ML database, namely the 100K and 1M. Note that these datasets are rather small for any outsourcing purpose. However, we only use these as examples to show our game theoretic framework and to outline the basic properties of it. In the SGD use case we used[2] a vector-based implementation in matlab, which mean whole vectors can be calculated independently instead of elements like in the Weighted Slope One case.

[1] Run on Xeon L5640@2.26GHz with 24GB RAM.
[2] Run on Xeon E7-4850@2GHz processor with 1 TB RAM.

This allows us to use much bigger database, so we used the Netflix dataset. Both of the implementation can be found at *https://github.com/ pidzso/Collaborative-filtering.*

Dataset	Users	Items	Ratings	Density	Missing
ML 100K	943	1682	100 000	0.0630	1 486 126
ML 1M	6040	3952	1 000 209	0.0419	22 869 871
Netflix	480 188	17 770	100 480 496	0.0118	8 432 460 264

Table 7: Datasets

The methodology is the following. We first fix K, B and θ and analyze the associated Normal form game. Then, we approximate the function $V_d(\sigma)$ using experimental results for the two example algorithms. We apply our framework to these use cases. Finally, we modify these algorithms' output to further minimize the client's cost.

5.1 Weighted Slope One Use Case

σ	T_{lin}	T_{org}
1.0000	102	102
0.1000	8.2	42.6
0.0100	1.0	37.2
0.0010	0.13	20.9
0.0001	0.017	3.1

(a) 100K dataset

σ	T_{lin}	T_{org}
1.000 000	2649	2649
0.100 000	221	1254
0.010 000	23	1090
0.001 000	3.0	1077
0.000 100	0.43	492
0.000 010	0.058	64
0.000 001	0.005	6.9

(b) 1M dataset

Table 8: Running Time (seconds)

Weighted Slope One has only two stages as shown in Section 2, where the second stage's computation relies on the first stage's output. In order to calculate $p_{u,i}$ (final output), one must calculate $\delta_{i,j}$ for all $j \in \mathbf{I}$ (first stage outputs) as it was argued above. To get the verification cost function (V_d) for this algorithm, we must measure the running time for different checking rates, shown in the column T_{org} in Table 8. Note that the running time for 100K in case of $\sigma = 0.000\,01$ and $0.000\,001$ are too small to be measured precisely, so that we skip them.

The construction of the verification cost out of these running times is basically a normalization: $V_d(\sigma) = \frac{T_{org}(\sigma)}{T_{org}(1)}$. Between the measured points, V_d is assumed to be linear to σ. It is shown in Figure 3 as *Original.* Note, that the lines here does not appear straight because of the logarithmic scale used to display them.

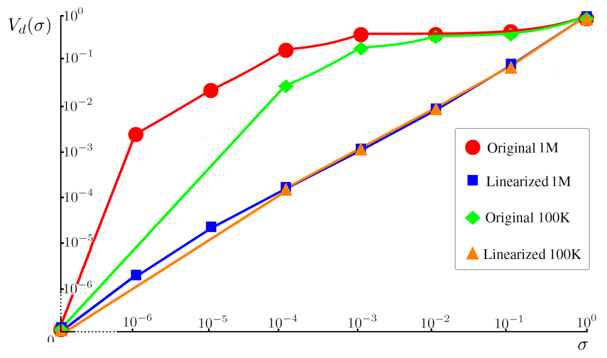

Fig. 3: V_d Cost Approximation

By using these approximations, we can minimize the client's cost $(w + V_d)$ with respect to the Inequality (11). Similar to the exhaustive search approach, we fix w first then determine σ and finally compute V_d. We present the results in Table 9.

θ	σ^*	w^*	$w^* + V_d^*$
0.001 00	0.003 02	0.011 83	0.252 57
0.000 10	0.037 11	0.003 84	0.384 49
0.000 01	0.207 43	0.033 83	0.520 99

(a) 100K Data Set

θ	σ^*	w^*	$w^* + V_d^*$
0.001 00	0.000 20	0.010 63	0.220 66
0.000 10	0.003 65	0.000 23	0.408 24
0.000 01	0.025 22	0.002 91	0.424 86

(b) 1M Data Set

Table 9: $(w^* + V_d^*)$ in Original Setting

This Table shows that despite of the very low fee w^* and checking rate σ^*, the final cost is much higher. This indicates, the majority of this cost is due to the verification cost V_d. So reducing it would have high impact on the final cost.

Linearization We modify the algorithm's output so that the output from the first stage (e.g. Δ) is also included in the final output. This means $\frac{I^2-I}{2}$ additional outputs, where I is the number of items. In case of 100K it is 1 413 721, while for 1M it is 7 807 176. This means +96% (+34%) growth in the final output size respectively, which could result in some communication overhead.

By using the extended output, the cheating and verification strategies must be slightly changed. Instead of only cheating on the predictions with ρ percent, the server cheats on Δ and on the predictions independently with ρ percent. The client verifies as follows.

1. It first chooses σ percent from the first stage's output and verifies them. If the verification passes, go to next step; otherwise stop.
2. It chooses σ percent of its output, and recalculates them using the already verified results from the previous stage.

With the above setup, the corresponding running time is shown in the T_{lin} column in Table 8, while the verification cost V_d is shown as *Linearized* in Fig. 3. Let K_1, K_2 be the output sizes of the two stages of Weighted Slope One. The detection rate is defined by Equation (12).

$$P_d(K_1, K_2, \sigma, \rho) = 1 - \left(\frac{\binom{(1-\rho)K_1}{\sigma K_1}}{\binom{K_1}{\sigma K_1}} \frac{\binom{(1-\rho)K_2}{\sigma K_2}}{\binom{K_2}{\sigma K_2}} \right) \tag{12}$$

Since the verification cost grows linearly with the size of the calculated data as seen in Fig. 3, we set $V_d^* = \sigma^*$. Table 10 shows the minimal cost and the corresponding parameters. Note that this was calculated based on Inequality (11) via exhaustive search on w.

θ	σ^*	w^*	$w^* + V_d^*$
0.001 00	0.002 75	0.000 35	0.003 10
0.000 10	0.019 52	0.003 40	0.022 92
0.000 01	0.115 61	0.029 76	0.145 37

(a) 100K Data Set

θ	σ^*	w^*	$w^* + V_d^*$
0.001 00	0.000 34	0.000 03	0.000 37
0.000 10	0.002 61	0.000 33	0.002 94
0.000 01	0.018 58	0.003 18	0.021 76

(b) 1M Data Set

Table 10: $(w^* + V_d^*)$ in Linearized Setting

These numbers suggest that, by linearizing the verification cost function via expanding the output set, for the same threshold cheating toleration rate, different parameters are optimal, i.e. the Equilibrium is changed. Importantly, for the same threshold cheating toleration rate, the client's cost $w^* + V_d^*$ is smaller than in the original setting (in Table 9).

If we compare the two tables in Table 9 or in Table 10, we see that the (normalized) cost for 1M is smaller than the cost for 100K for the same cheating toleration rate after different normalization. After taking into the normalization factor, the cost for 1M is certainly larger than that of 100K.

5.2 SGD Use Case

SGD has more iterations (stages) before producing its output [2], and it has many parameters as shown in Section 2. These parameters can be optimized dataset-wise, but we set them in an independent manner within reasonable boundaries. In more detail, the regularization parameter is $\gamma = 0.001$, the learning rate is $\epsilon = \frac{50}{N}$, the factor size is $c = 10$ and the iteration number is $\upsilon = 100$. SGD algorithm's pseudocode is shown in Algorithm 1, where t_u and s_i are the factor vectors, Ψ is the set of known ratings and RMSE is the Root-Mean-Square-Error: $\sqrt{\frac{\sum(r_{u,i} - \hat{r}_{u,i})^2}{N}}$.

In each iteration, the factor vectors t_u and s_i are updated by many (different) existing rating values in Ψ. For simplicity, we can define the output of each iteration to include the following information:

Algorithm 1 SGD Sketch

```
1: Initializing $t_u$ and $s_i$ for all $u$ and $i$
2: for $j = 1 \dots \upsilon$ do
3:     Permute $\Psi$
4:     for all $p_{u,i} \in \Psi$ do
5:         Update $t_u$ and $s_i$
6:     end for
7:     Calculate RMSE
8: end for
```

- A user factor matrix formed by t_u for all u.
- An item factor matrix formed by s_i for all i.
- The initial values of t_u and s_i for all u and i.
- The permutation done to Ψ.

As a result, there are only $K = \upsilon = 100$ verifiable outputs independently which dataset was used.

Table 11 shows the running time of Algorithm 1. T_{all} is the running time for the whole algorithm, T_{ini} shows the initialization running time, the column T_{iter} shows the average time for one iteration and T_{rmse} shows average the running time to compute RMSE for one iteration.

	T_{all}	T_{ini}	T_{iter}	T_{rmse}
Netflix	276 030	30	2700	60

Table 11: Running Time (seconds)

Just as in the Weighted Slope One example, now we construct the verification function $V_d(\sigma)$ using the measured runnng times above. We include the RMSE calculation as part of the server's computation, because it must know whether the actual iteration's result has converged or not. However, the client does not need to verify that, so it is not included in the client's verification time. This means $V_d(1) < 1$. The overall running time (i.e. $\upsilon(T_{iter} + T_{rmse}) + T_{ini}$) is 76.675 hours, which is the normalizer factor. Using these, the computational cost for one iteration is $\frac{T_{iter}}{\upsilon(T_{iter}+T_{rmse})+T_{ini}}$, which means $V_d(\sigma) = \frac{T_{iter}}{\upsilon(T_{iter}+T_{rmse})+T_{ini}}\sigma K$, which means $V_d \approx 0.978\sigma$. The minimum costs are computed via exhaustive search on w and shown in Table 12 when cheating on 2, 5 and 10 outputs is tolerated.

Table 12 shows that due to the small output size, a significant part of the results must be recalculated, which means high verification cost V_d. To avoid this, we shall expand the output with intermediate results just as in the previous example.

Linearization Instead of the factor matrices (which are the results at the end of each iteration), we define the outputs to include: all the updates (i.e. the

θK	$\sigma^* K$	w^*	$w^* + V_d^*$
2	63	0.155 47	0.771 61
5	36	0.112 69	0.464 77
10	23	0.067 67	0.292 61

Table 12: Normalized Costs

corresponding values of t_u and s_i for all the N update) in the iteration with the auxiliary information Ψ and the initial values of t_u and s_i. Since we have υ iteration, in total there will be υN verifiable output results. Correspondingly, we adjust the cheating and verification strategies as follows. In the output of each iteration, the server cheats on with ρ percent of the results. The verification strategy is very simple: the client chooses σ percent of the updating in each iteration to verify. As such, the detection rate is defined by Equation (13).

$$P_d(N, \upsilon, \sigma, \rho) = 1 - \left(\frac{\binom{(1-\rho)N}{\sigma N}}{\binom{N}{\sigma N}} \right)^{\upsilon} \tag{13}$$

In this setting, the output size is $K = \upsilon N$. This means 100 billion vectors. This is only 19% more verifiable results respectively than the number of missing predictions, however, one output here is vector with dimension of c instead of a value, so since $c = 10$ this actually means 190%. If the communication channel's capacity for transmitting information is not sufficiently high, it can jeopardize the whole outsourcing process.

Lets define T_{upd} as the time required for an update using one sample. We can set it linearly because this is the basic block of the algorithm, so $T_{upd} = \frac{T_{iter}}{N}$. Now we can redefine the verification cost function in this new setting as follows: $V_d(\sigma) = \frac{T_{upd}}{\upsilon(T_{iter}+T_{rmse})+T_{ini}}\sigma K$, which is actually the same $V_d(\sigma) \approx 0.978\sigma$. The only difference is the number of verifiable outputs. As it was before, the cost minimization for Inequality (11) is done by exhaustive search on w where $w \in \{0.000\,001, 0.000\,002, \dots\}$. The results are shown in Table 13.

θ	σ^*	w^*	$w^* + V_d^*$
0.000 010 00	0.000 014	0.000 001	0.000 014
0.000 005 00	0.000 026	0.000 002	0.000 028
0.000 002 50	0.000 049	0.000 004	0.000 052
0.000 001 00	0.000 115	0.000 010	0.000 122
0.000 000 50	0.000 216	0.000 019	0.000 231
0.000 000 25	0.000 404	0.000 039	0.000 434
0.000 000 10	0.000 603	0.000 001	0.000 590

Table 13: Normalized Costs

It is clear that the integration of all the possible intermediate results into the final output has resulted in lower final costs. The table suggest that for high

(> 0.0001) and low ($< 0.000\,001$) tolerated cheating rate, even the minimal fee ensures the honest behavior, while around $\theta = 0.000\,01$ the Equilibrium changes.

5.3 Comparison

After studying these two use cases, a natural question is which is more cost (time) efficient. For this purpose, we use the ML 1M dataset. The linearized Weighted Slope One algorithm running time is 2649 second as it was seen in Table 8, while the SGD's running time on this dataset is $T_{iter} = 2.53$ and $T_{rmse} = 0.1819$ seconds. This mean, the total time with 100 iteration is 271.19 seconds. Table 14 summarizes the time requirements for different tolerated cheating rates. The **WSO** column represents the linearized Weighted Slope One case where the numbers are generated via multiplying w and σ from Table 10 with the total running time (e.g. with 2649). The **SGD** column shows the linearized SGD case, where the numbers are calculated as follows.

- The checking rate σ is converted to time consumption by multiplying it with $\upsilon T_{iter} = 253$.
- The payment is $W = 1 + w$, where 1 is the cost of the calculation. So the fee w can be viewed as a portion of the calculation cost. By multiplying it with the total cost 271, we get the 'time value' of it.
- We modify the Inequality (11) by replacing $\min_w[w + V_d]$ to $\min_w[271w + 253V_d]$ and minimize it via exhaustive search on w.

θ	**WSO**	**SGD**
0.001 00	0.978 39	0.032 05
0.000 10	7.787 45	0.260 08
0.000 01	57.6475	1.362 74

Table 14: Comparison of the two Use Cases

The numbers ($2649(w + V_d)$ for **WSO** and $271w + 253V_d$ for **SGD**) shows, that SGD outperforms Weighted Slope One in term of the client's cost (expressed by the time requirements).

6 Conclusion

Nowadays outsourcing computation to cloud service providers is very common. Guaranteeing the correctness of these calculations is usually done by verification mechanisms. In this paper we proposed a game theoretic framework which can be applied on a top of a verification mechanism to maximize the client's payoff while ensuring the correctness of the results by providing the right parameters. We chose two representative recommender algorithm and showed that expanding the algorithm's output with intermediate results archives less verification cost despite of the increased output size. In the future, there are many possible directions to extend this research. For example, the leader and follower roles can be switched, so the server will make the offer w what it want to receive in exchange for the results. Making the game sequential is also a possible way to go. Introducing reputation or using punishment is also reasonable, especially in the sequential game scenario.

References

1. Boxiang Dong, Ruilin Liu, and Hui Wendy Wang. Result integrity verification of outsourced frequent itemset mining. In *Data and Applications Security and Privacy XXVII*, pages 258–265. Springer, 2013.
2. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, pages 30–37, 2009.
3. Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *SDM*, volume 5, pages 1–5. SIAM, 2005.
4. Ruilin Liu, Philippos Mordohai, Wendy Hui Wang, Hui Xiong, Ruilin Liu, Hui Wendy Wang, Philippos Mordohai, and Hui Xiong. Integrity verification of k-means clustering outsourced to infrastructure as a service (iaas) providers. In *SDM*, pages 632–640. SIAM, 2013.
5. Lacra Pavel. Basics of game theory. In *Game Theory for Control of Optical Networks*, pages 11–26. Springer, 2012.
6. Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.
7. Qiang Tang, Balázs Pejó, and Husen Wang. Protect both integrity and confidentiality in outsourcing collaborative filtering computations. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*. IEEE, 2016.
8. Jaideep Vaidya, Ibrahim Yakut, and Anirban Basu. Efficient integrity verification for outsourced collaborative filtering. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 560–569. IEEE, 2014.
9. Wai Kit Wong, David W Cheung, Edward Hung, Ben Kao, and Nikos Mamoulis. Security in outsourcing of association rule mining. In *Proceedings of the 33rd international conference on Very large data bases*, pages 111–122. VLDB Endowment, 2007.
10. Wai Kit Wong, David W Cheung, Edward Hung, Ben Kao, and Nikos Mamoulis. An audit environment for outsourcing of frequent itemset mining. *Proceedings of the VLDB Endowment*, 2(1):1162–1173, 2009.